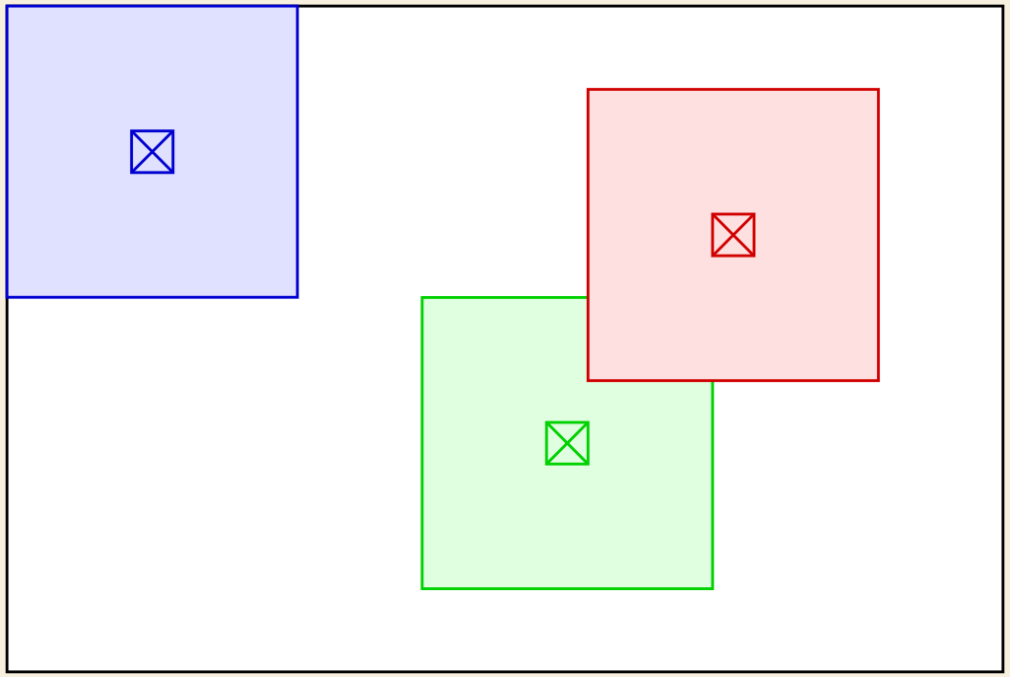
СИСТЕМЫ ТЕХНИЧЕСКОГО ЗРЕНИЯ

БИНАРИЗАЦИЯ ИЗОБРАЖЕНИЙ. ФИЛЬТРАЦИЯ

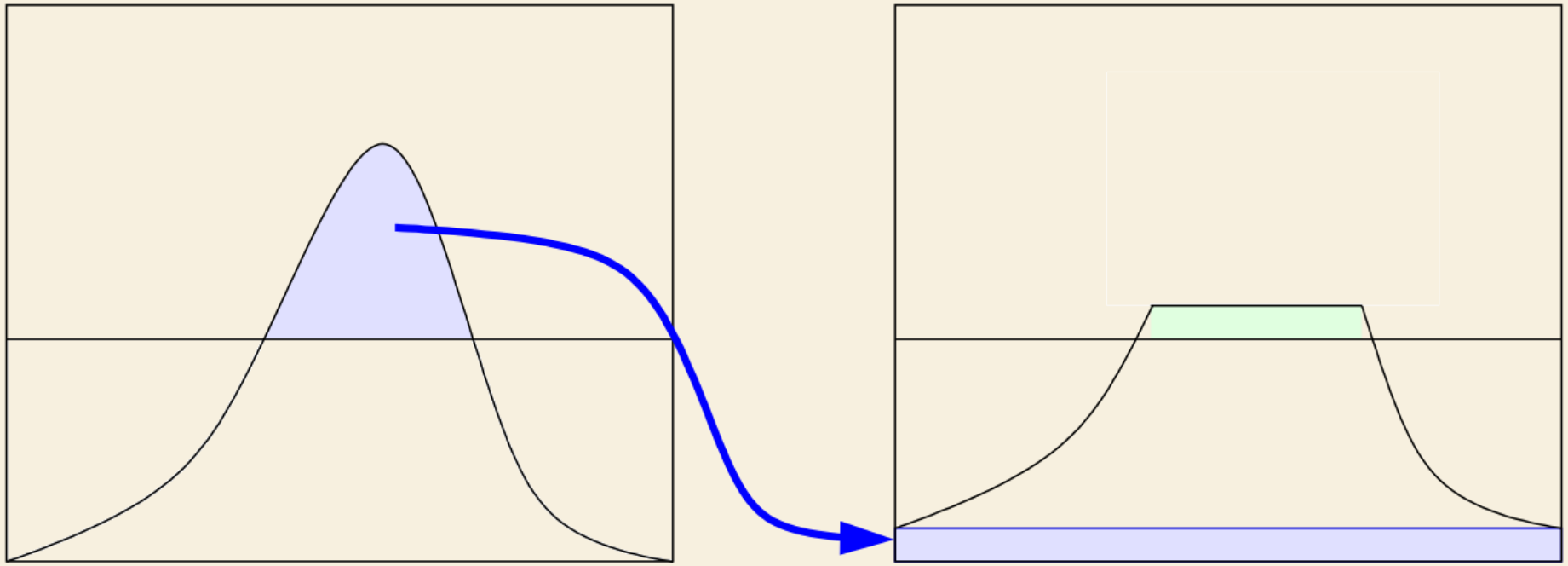
АДАПТИВНАЯ БИНАРИЗАЦИЯ



- Одноканальное изображение
- Мы будем вычислять новое значение яркости в пикселе, строя гистограмму по его окрестности
- При это среднюю яркость мы «подтягиваем» к середине диапазона
- *AHE* – Adaptive histogram equalization

УЧЁТ КОНТРАСТА

- Мы можем обрезать выбросы на гистограмме
- Их можно учесть, сместив функцию распределения, «размазав» выброс по всей области
- *CLAHE* – Contrast limited AHE



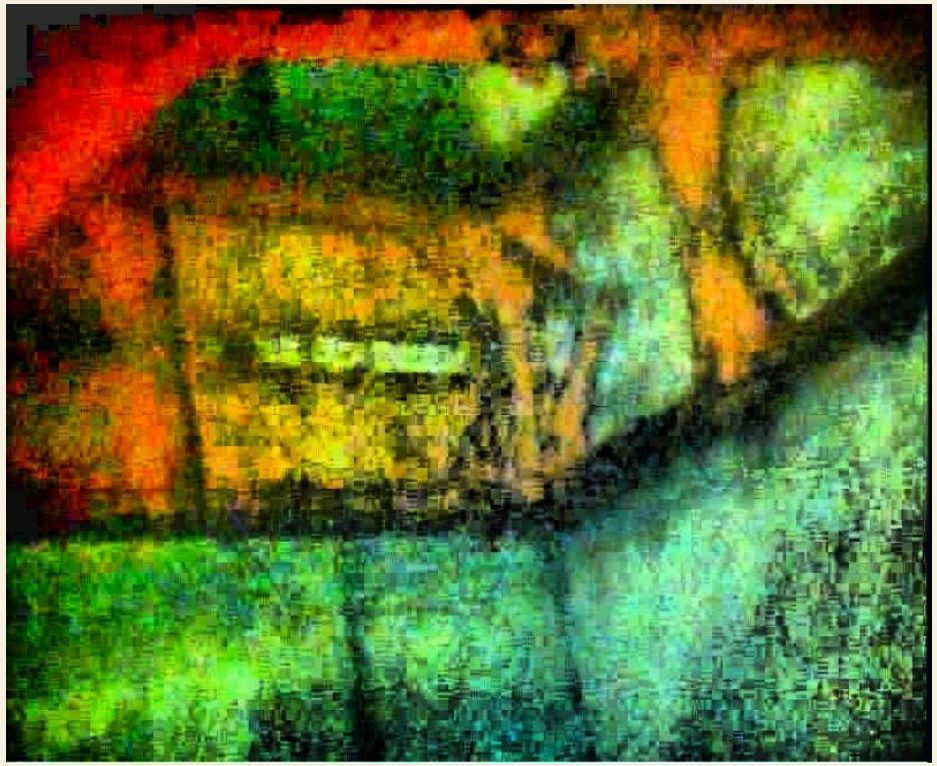
```cpp
#include <iostream>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgcodecs.hpp>

using namespace std;
using namespace cv;

int main()
{
    Mat src, dst;
    string fileName = "Picture1.png";
    src = imread(fileName, IMREAD_COLOR);
    if (src.empty())
        return -1;

    cvtColor(src, src, CV_BGR2HSV);
    Mat channels[3];
    split(src,channels);


    Ptr<CLAHE> pClahe = createCLAHE(80, Size(5,5));
    cout << pClahe->getTilesGridSize() << endl;
    cout << pClahe->getClipLimit() << endl;
    pClahe->apply(channels[2],channels[2]);
    merge(channels, 3, dst);

    cvtColor(dst,dst, CV_HSV2BGR);
    imshow("Result", dst);
    waitKey(0);

    return 0;
}
```

В OPENCV

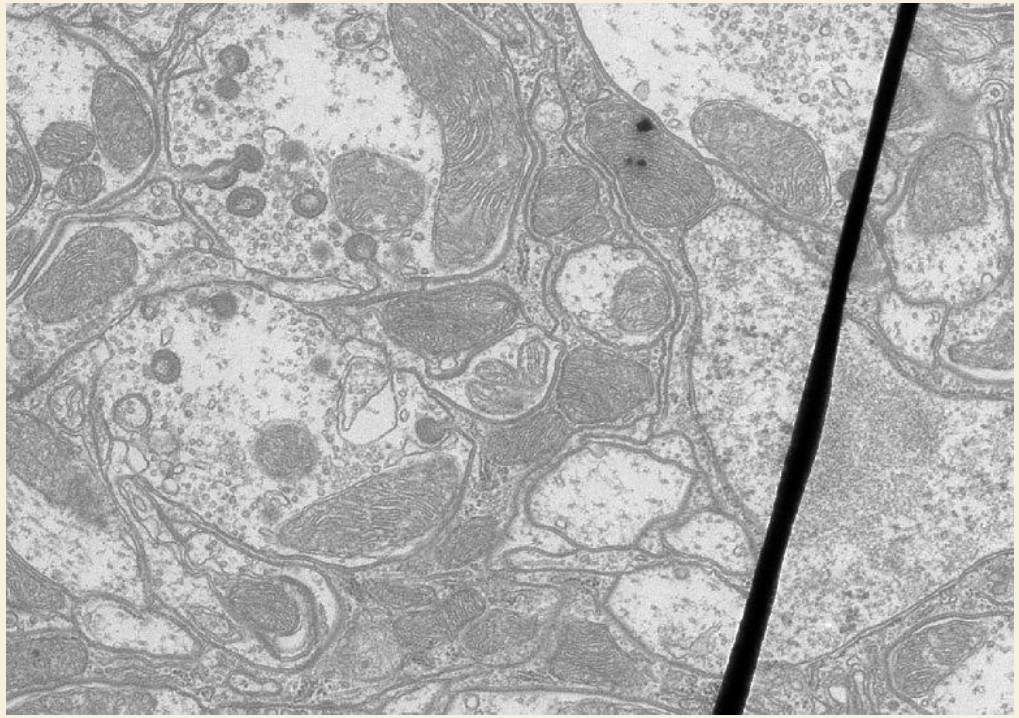
У метода createCLAHE 2 параметра:

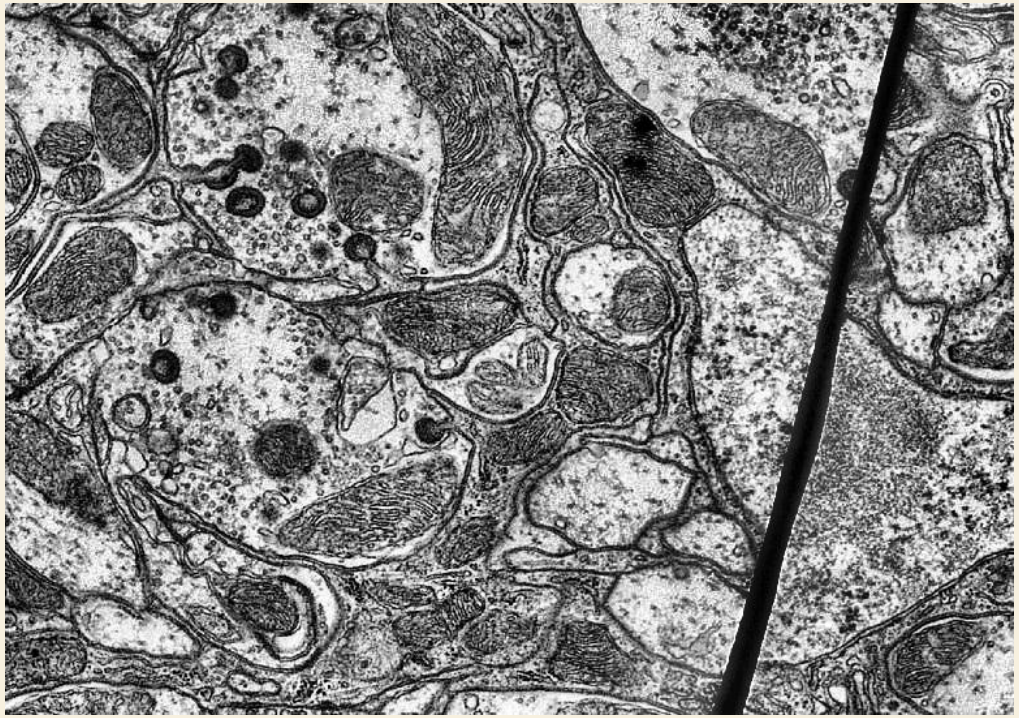
1. Ширина зоны до обрезки (по умолчанию 40)
2. Размер области, по которой вычисляется гистограмма (по умолчанию 8x8)

И ЕЩЁ РЕЗУЛЬТАТ



РЕЗУЛЬТАТ

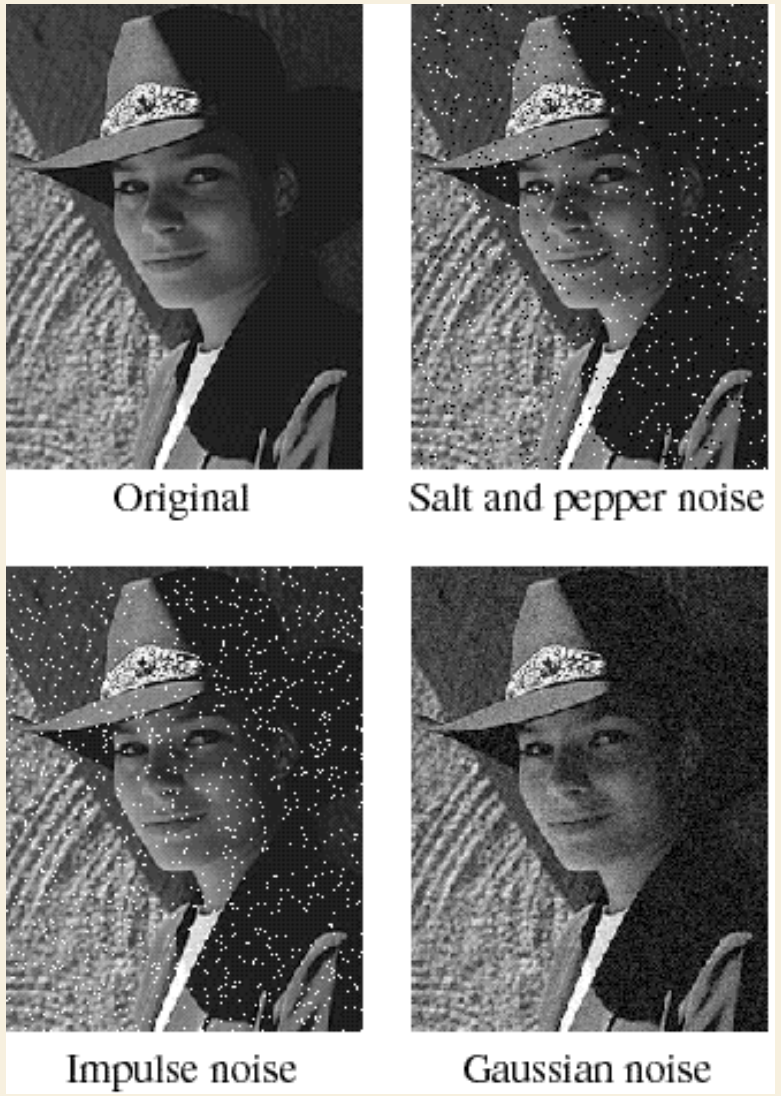




ШУМОПОДАВЛЕНИЕ

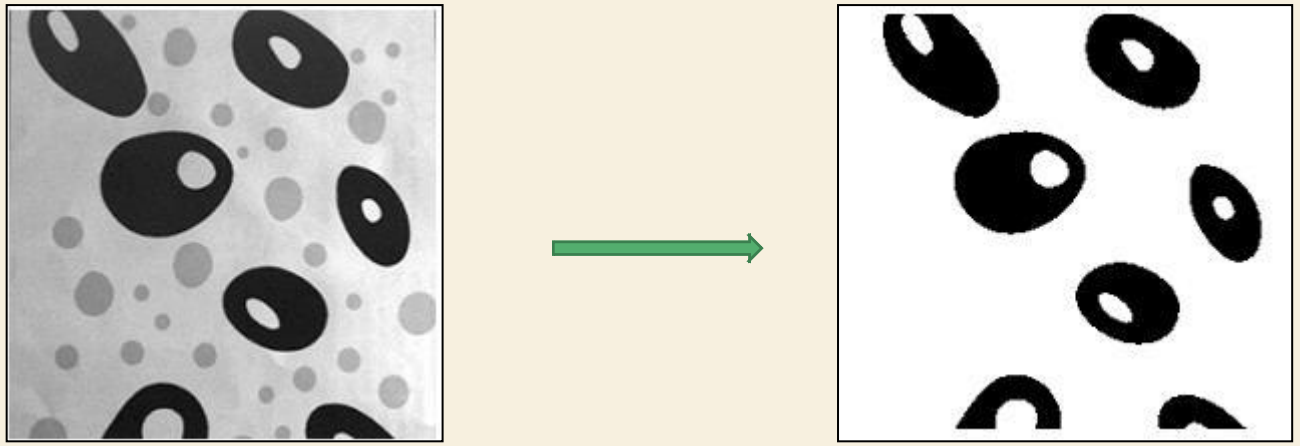
Соль и перец:
случайные черные и белые пиксели

- **Импульсный:**
случайные белые пиксели
- **Гауссов: колебания**
яркости, распределенные по нормальному закону



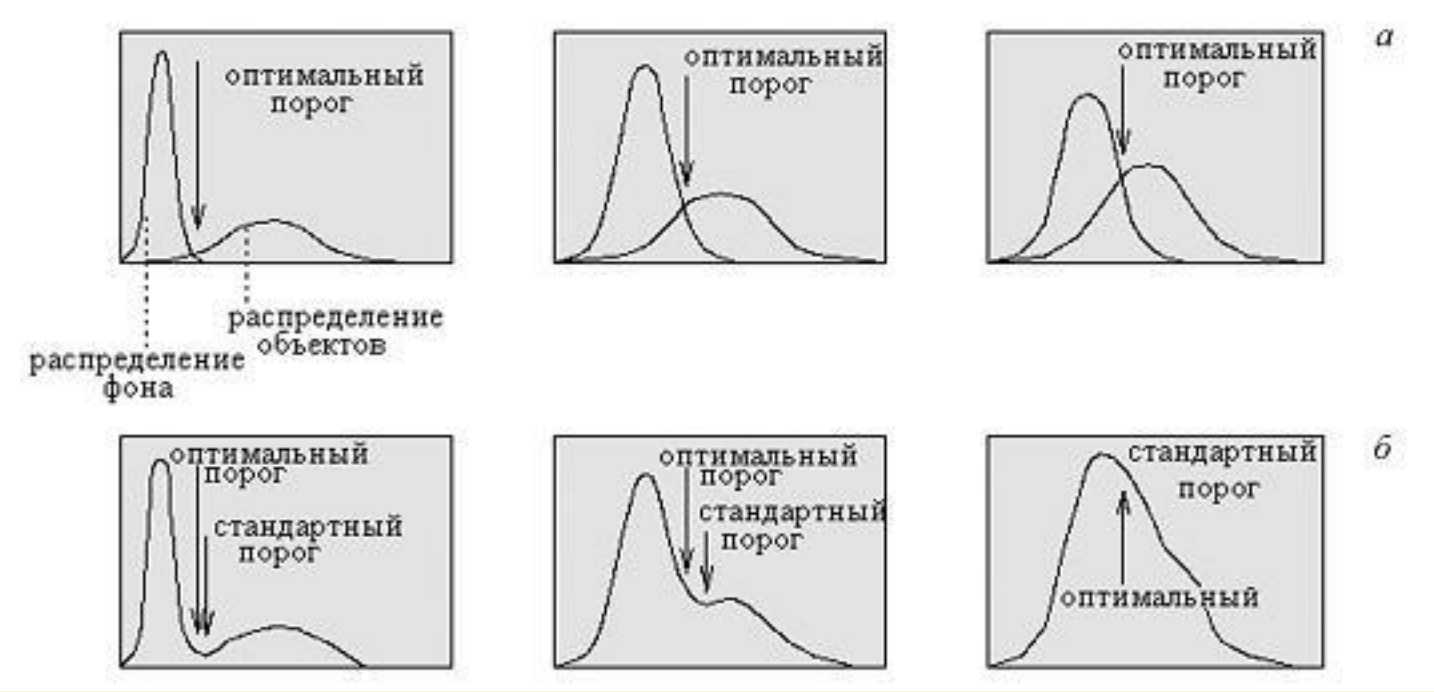
БИНАРНОЕ ИЗОБРАЖЕНИЕ

- Уменьшение объёма информации
- Упрощение её обработки
- Как её произвести?



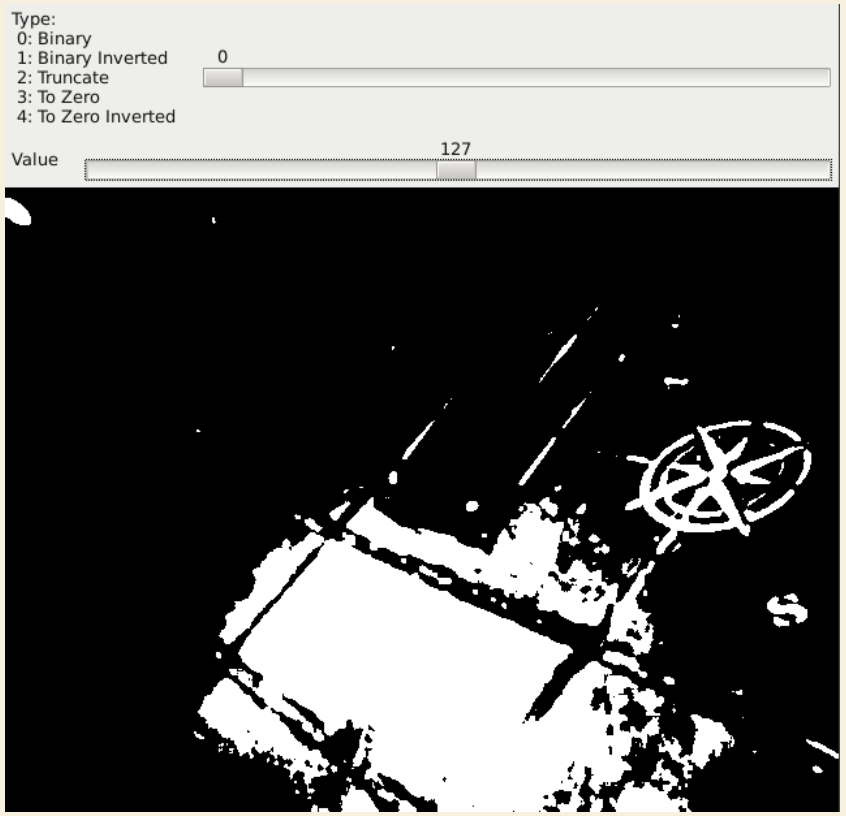
БИНАРНОЕ ИЗОБРАЖЕНИЕ

- На основе обработки гистограмы яркостей
- Выбор конкретного положения?



OPENCV_BINARIZATION

ОРИГИНАЛ



БИНАРИЗАЦИЯ

```
#include "opencv2/imgproc.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include <iostream>
using namespace cv;
using std::cout;
int threshold_value = 0;
int threshold_type = 3;
int const max_value = 255;
int const max_type = 4;
int const max_binary_value = 255;
Mat src, src_gray, dst;
const char* window_name = "Threshold Demo";
const char* trackbar_type = "Type: \n 0: Binary \n 1: Binary Inverted \n 2: Truncate \n 3: To Zero \n 4: To Zer
o Inverted";
const char* trackbar_value = "Value";
static void Threshold_Demo( int, void* )
{
    /* 0: Binary
     1: Binary Inverted
     2: Threshold Truncated
     3: Threshold to Zero
     4: Threshold to Zero Inverted
    */
    threshold( src_gray, dst, threshold_value, max_binary_value, threshold_type );
    imshow( window_name, dst );
}
```

MAIN()

```
int main( int argc, char** argv )
{
    String imageName("image.png"); // by default
    src = imread( imageName, IMREAD_COLOR ); // Load an image
    if (src.empty())
    {
        cout << "Cannot read the image: " << imageName << std::endl;
        return -1;
    }
    cvtColor( src, src_gray, COLOR_BGR2GRAY ); // Convert the image to Gray
    namedWindow( window_name, WINDOW_AUTOSIZE ); // Create a window to display results
    createTrackbar( trackbar_type,
                    window_name, &threshold_type,
                    max_type, Threshold_Demo ); // Create a Trackbar to choose type of Threshold
    createTrackbar( trackbar_value,
                    window_name, &threshold_value,
                    max_value, Threshold_Demo ); // Create a Trackbar to choose Threshold value
    Threshold_Demo( 0, 0 ); // Call the function to initialize
    waitKey();
    return 0;
}
```

БИНАРНОЕ ИЗОБРАЖЕНИЕ

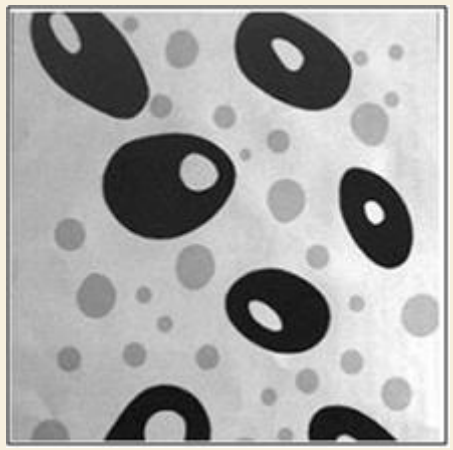
- Критерий Отсу (Оцу)
- Вычисляются математическое ожидание μ и вероятность ω для каждого значения яркости
- Вычисляются их суммы и ищется минимум

$$\sigma_b^2(t) = \sigma^2 - \sigma_\omega^2(t) = \omega_1(t)\omega_2(t)[\mu_1(t) - \mu_2(t)]^2$$

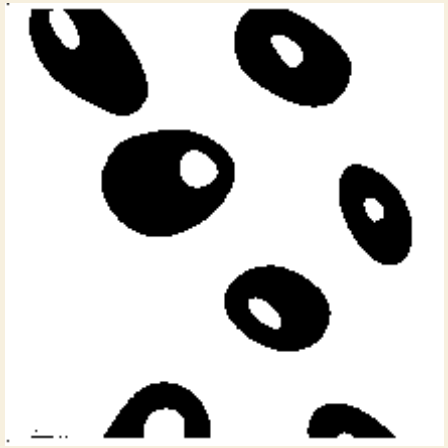
- Более подробное описание https://docs.opencv.org/trunk/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57

БИНАРНОЕ ИЗОБРАЖЕНИЕ

ОРИГИНАЛ

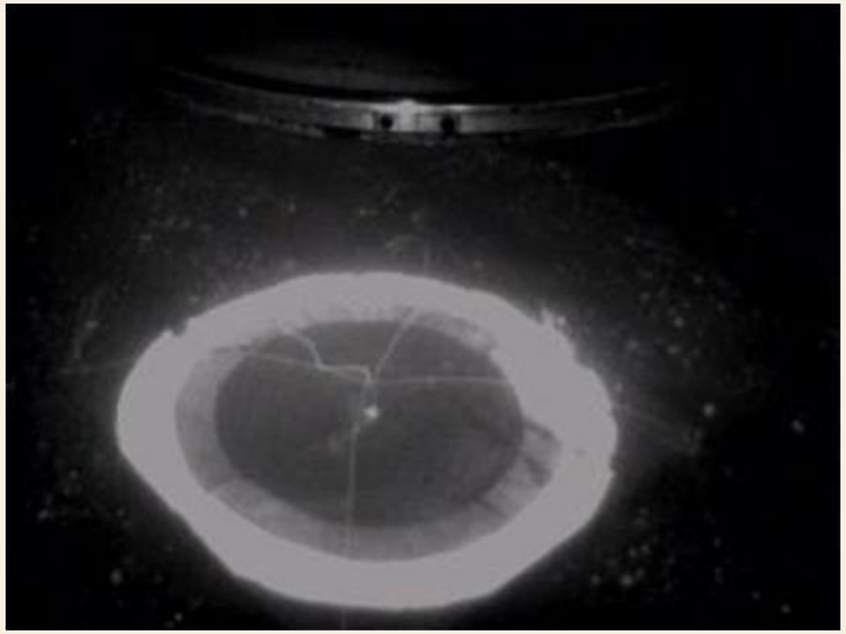


БИНАРИЗАЦИЯ **THRESH_BINARY**

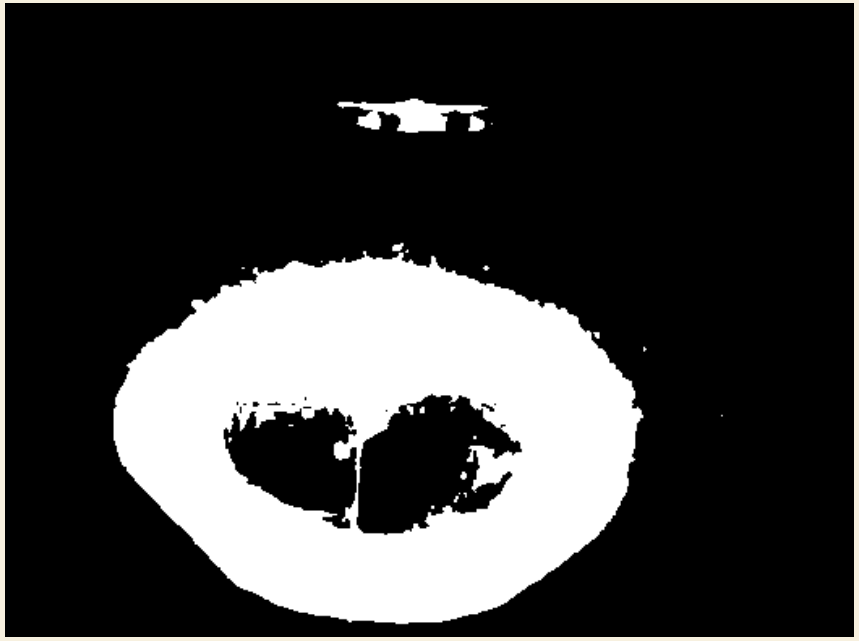


БИНАРНОЕ ИЗОБРАЖЕНИЕ

ОРИГИНАЛ



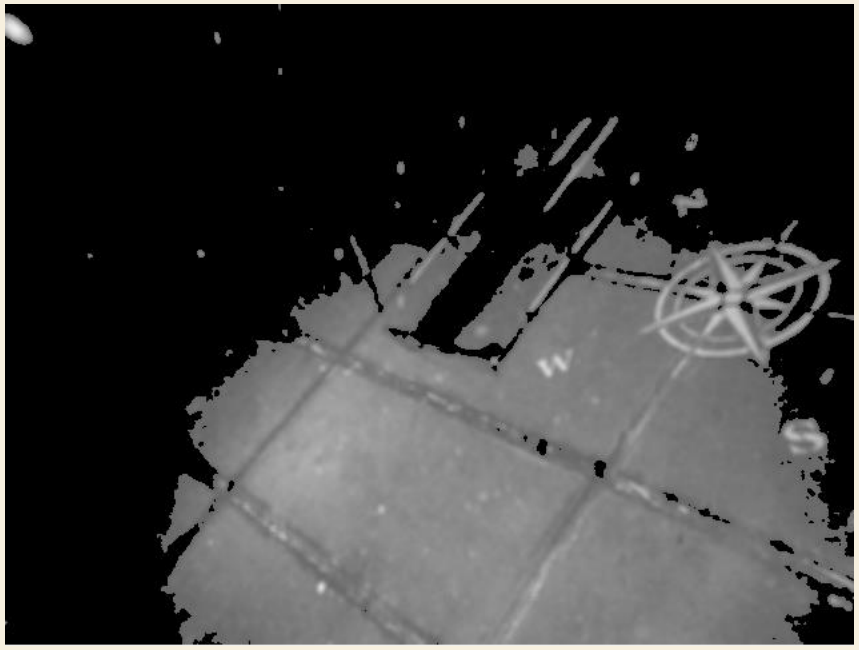
БИНАРИЗАЦИЯ THRESH_BINARY



БИНАРНОЕ ИЗОБРАЖЕНИЕ: А ЕСЛИ КОНТРАСТНОСТЬ НИЖЕ?

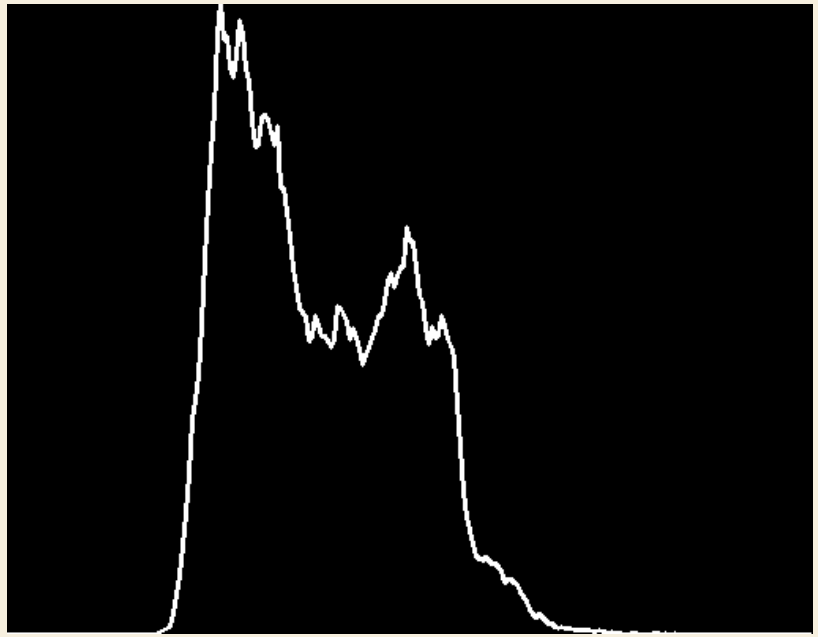
ОРИГИНАЛ

БИНАРИЗАЦИЯ TO_ZERO

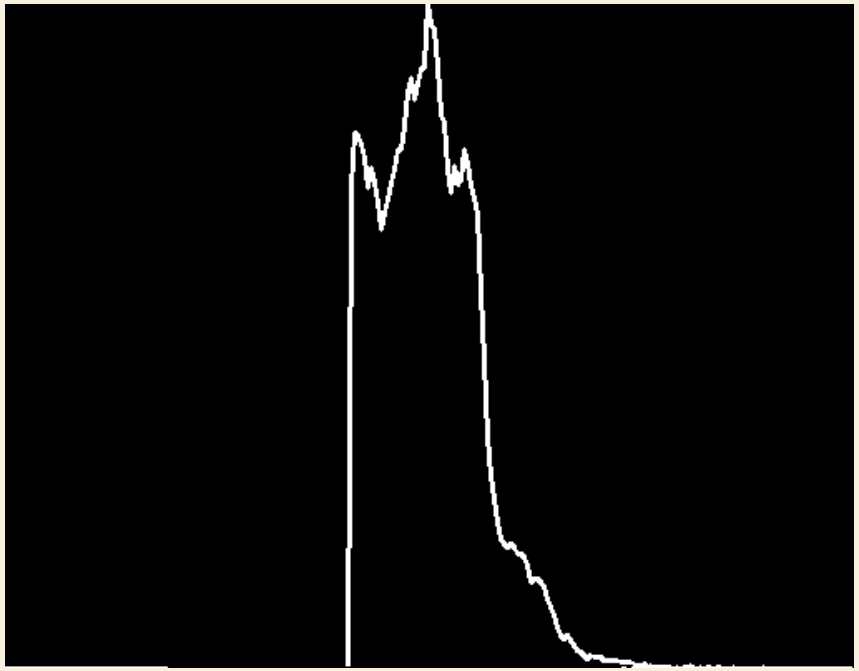


БИНАРНОЕ ИЗОБРАЖЕНИЕ: А ЕСЛИ КОНТРАСТНОСТЬ НИЖЕ?

ОРИГИНАЛ



БИНАРИЗАЦИЯ TO_ZERO



АДАПТИВНАЯ БИНАРИЗАЦИЯ

- Попробуем бинаризовать не всё изображение, а его отдельные области
- Точнее, будем сравнивать пиксель с его окружением
- Если он ярче порогового значения – 1, иначе – 0
- Как выбрать область?

АДАПТИВНАЯ БИНАРИЗАЦИЯ

MEAN 5X5

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

GAUSS 5X5

1	2	3	2	1
2	4	6	4	2
3	6	9	6	3
2	4	6	4	2
1	2	3	2	1

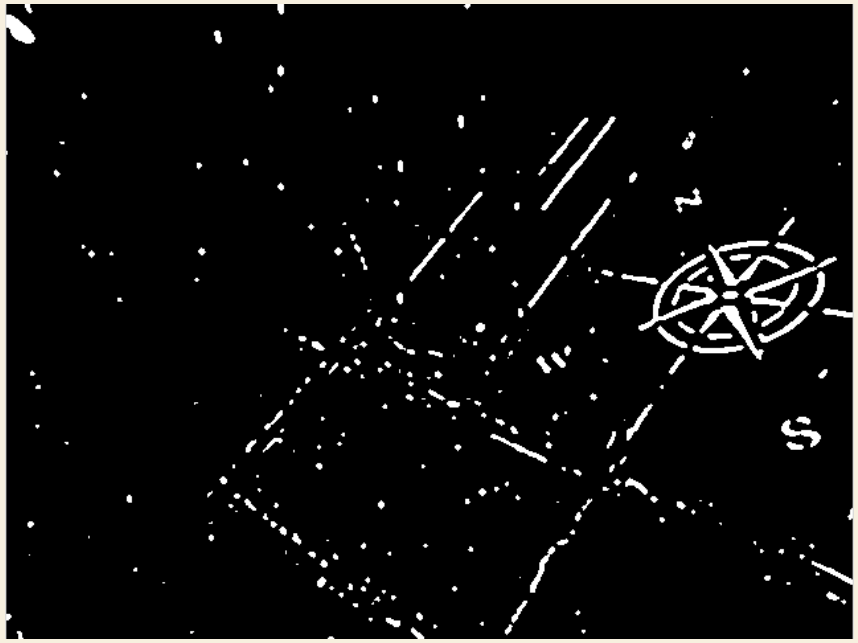
АДАПТИВНАЯ БИНАРИЗАЦИЯ

- Суммируем по области размером (size x size)
- При суммировании используем веса
- Делим на сумму весов

АДАПТИВНАЯ БИНАРИЗАЦИЯ

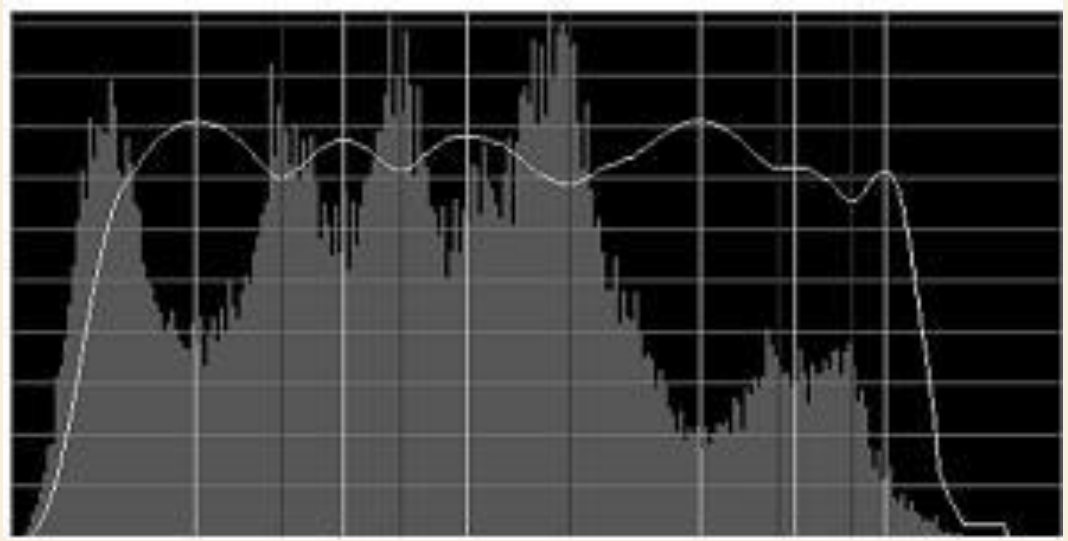
ОРИГИНАЛ

БИНАРИЗАЦИЯ **MEAN**



СЕГМЕНТИРОВАННОЕ ИЗОБРАЖЕНИЕ

- Для многомодового случая нет универсального подхода



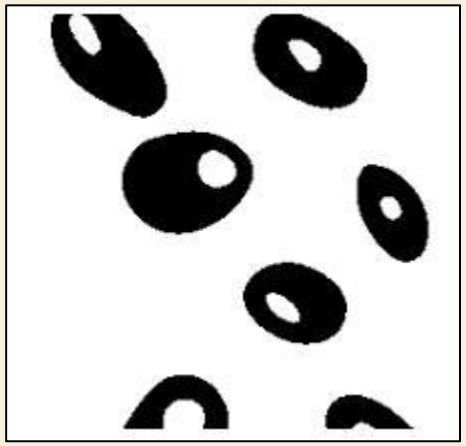
СЕГМЕНТИРОВАННОЕ ИЗОБРАЖЕНИЕ

БИНАРНОЕ ИЗОБРАЖЕНИЕ

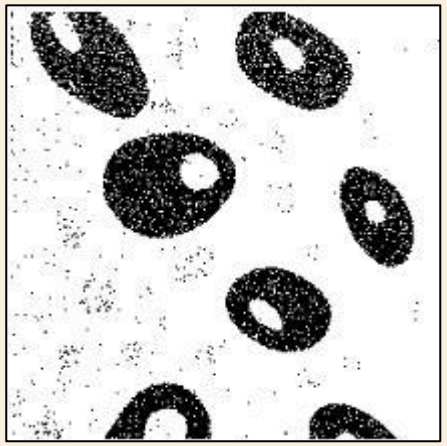
Модель шума «соль-перец»
вероятности перехода

$Im[x,y] \rightarrow Im'[x,y]$	$Im'[x,y]=1$	$Im'[x,y]=0$
$Im[x,y]=1$	$1-p$	p
$Im[x,y]=0$	q	$1-q$

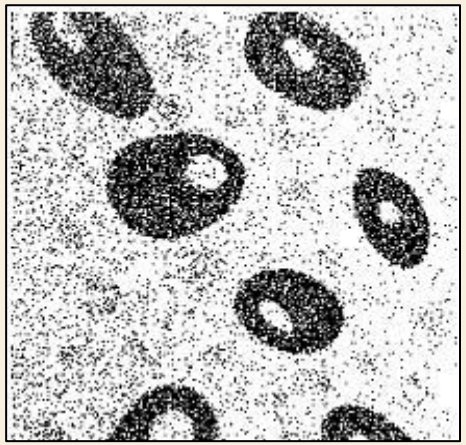
БИНАРНОЕ ИЗОБРАЖЕНИЕ



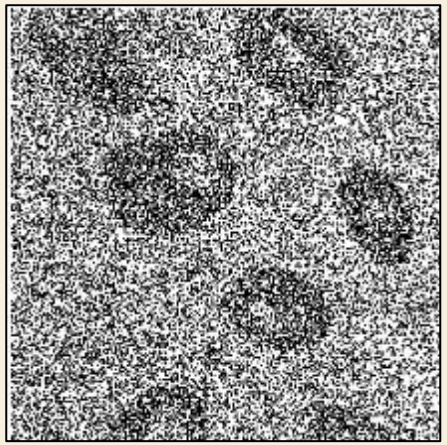
исходное



p=0.1
q=0.1



p=0.25
q=0.25



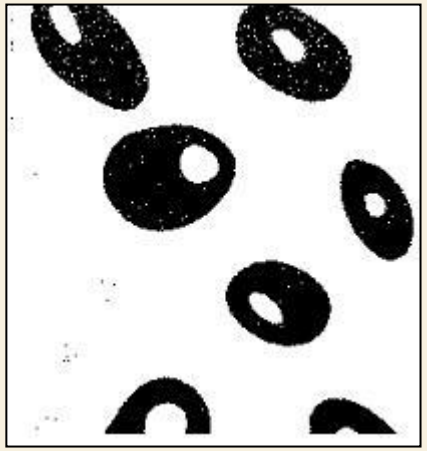
p=0.45
q=0.45

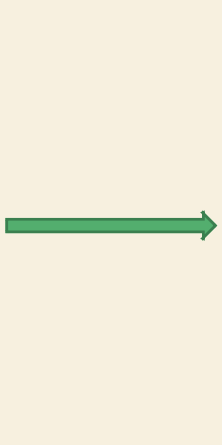
БИНАРНОЕ ИЗОБРАЖЕНИЕ

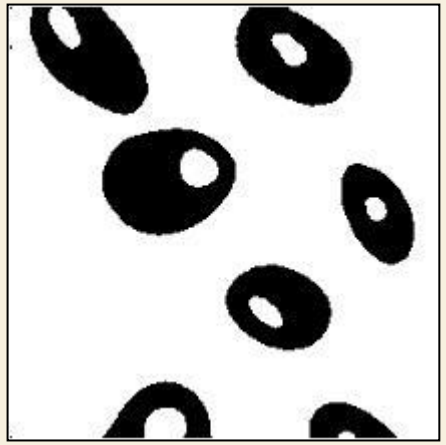
- Медианный фильтр
- Считаем количество 0 и 1 в *n*-окрестности

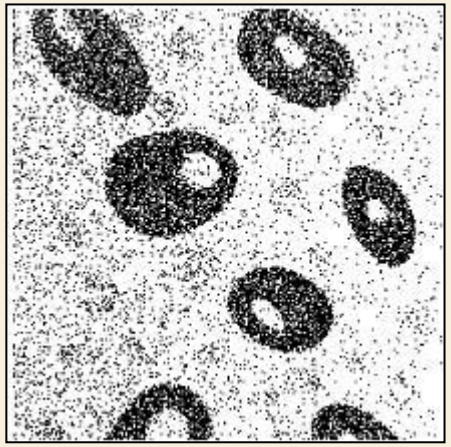
1	1	0
1	0	1
1	1	0

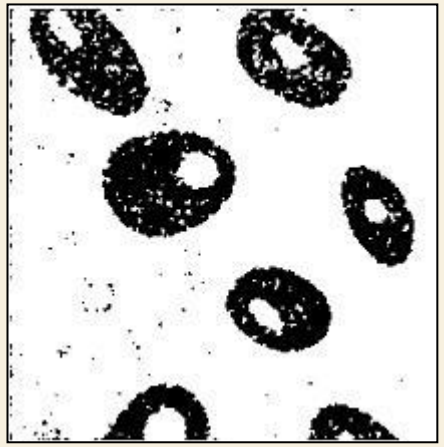
- Единиц > нулей => 1





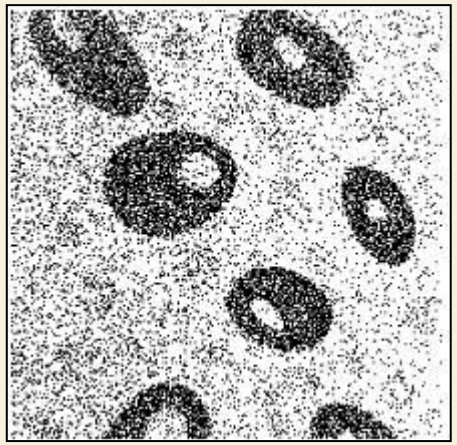


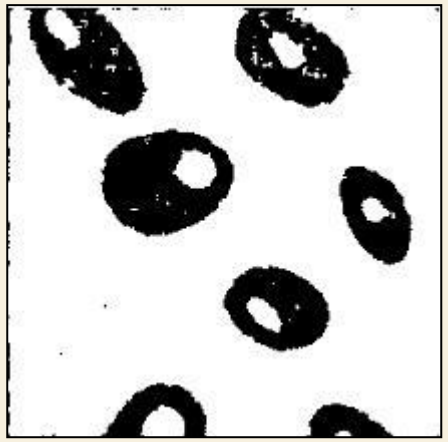


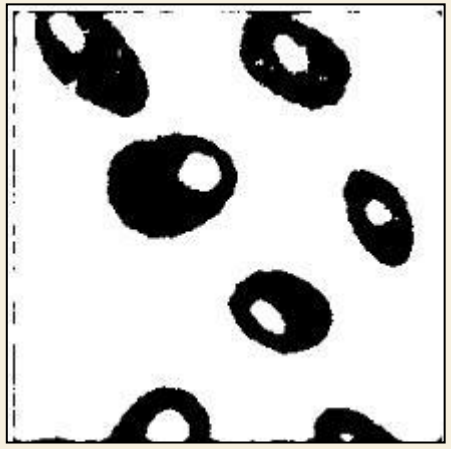


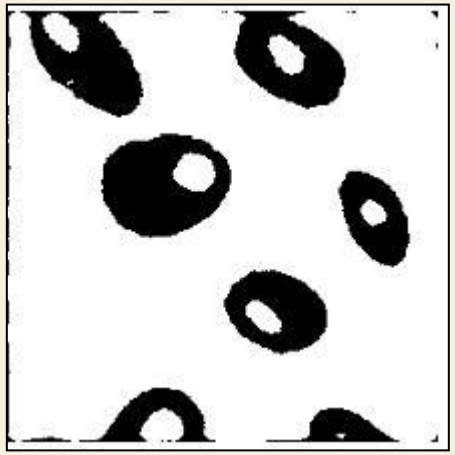
БИНАРНОЕ ИЗОБРАЖЕНИЕ

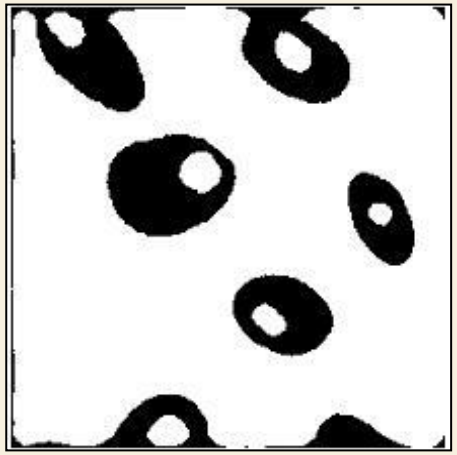
Результат зависит от зашумлённости











БИНАРНОЕ ИЗОБРАЖЕНИЕ

Результат зависит от размера апертуры

БИНАРНОЕ ИЗОБРАЖЕНИЕ

- ранговый фильтр – больше заданного порога
- разные пороги для нулей и единиц!
- Он же процентильный

1	1	0
1	0	1
1	1	0

- Единиц 6, нулей 3
- Ранг 6 – 1, ранг 7 – 0
- Нужен, если мы имеем априорную информацию

ГАУССОВ ШУМ

- Модель аддитивного шума

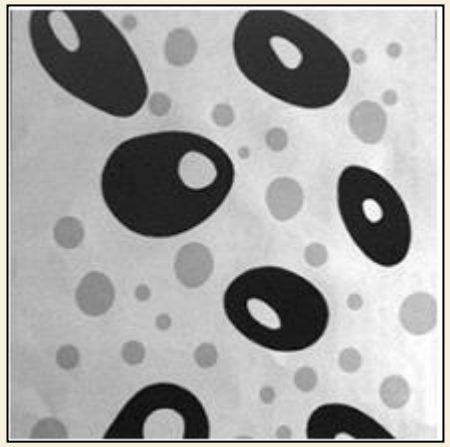
$$Im'[x,y]=Im[x,y]+R(x,y)$$

- Частный случай – гауссов шум

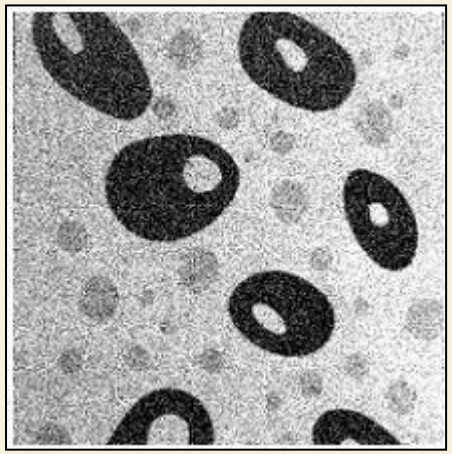
$$Im'[x,y]=Im[x,y]+N(o,\sigma),$$

- Мат.модель: сумма множества независимых факторов
- Подходит при маленьких дисперсиях
- Предположения: независимость, нулевое матожидание

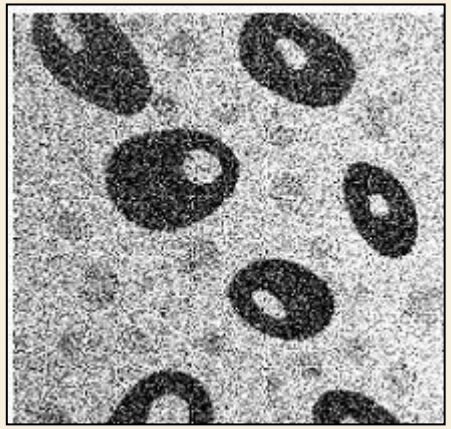
ГАУССОВ ШУМ



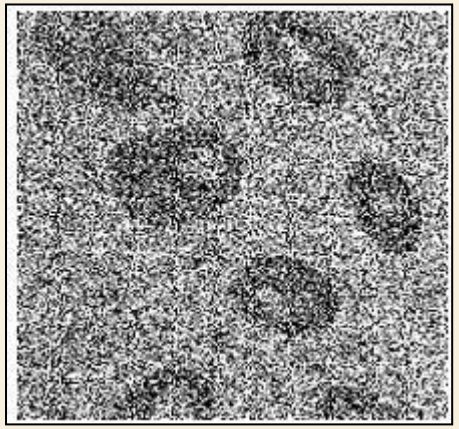
исходное



σ=40



σ=80



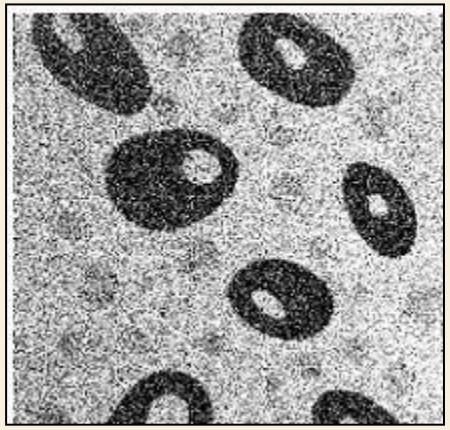
σ=300

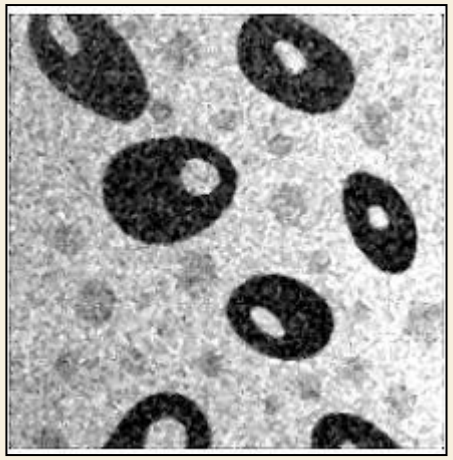
ГАУССОВ ШУМ

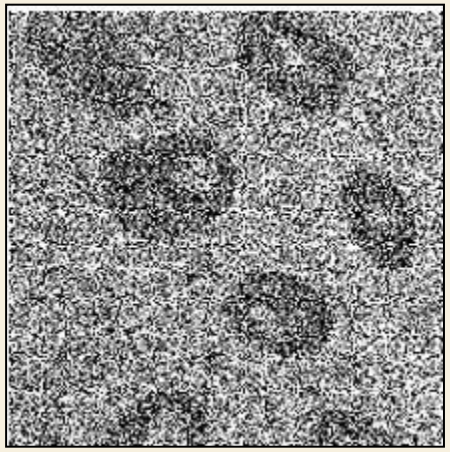
- Медианный фильтр
- Упорядочиваем точки в *n*-окрестности

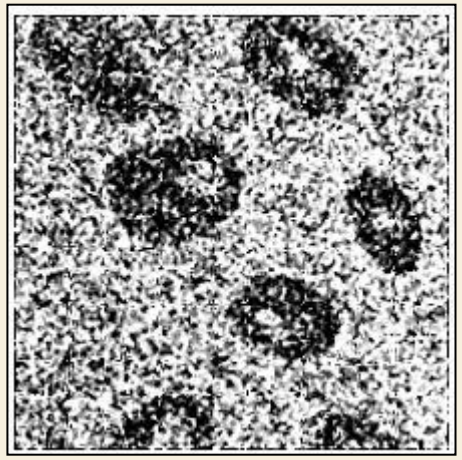
173	164	170
150	176	169
168	182	166

- (150, 164, 166, 168, 169, 170, 173, 176, 182) – в середине списка



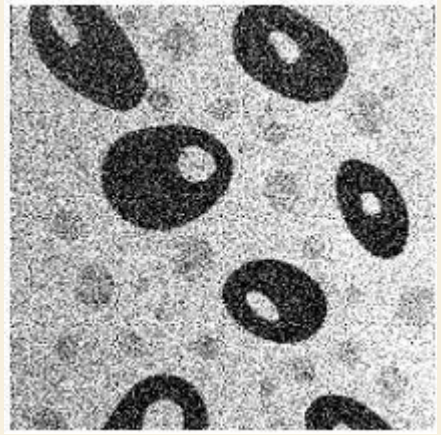


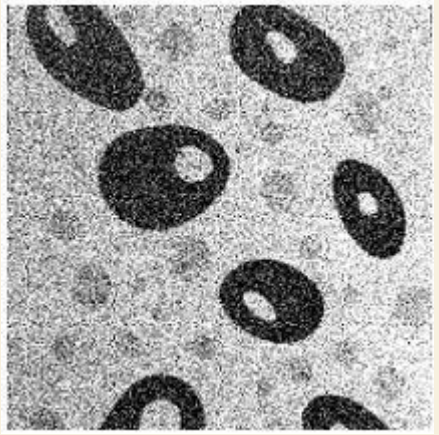


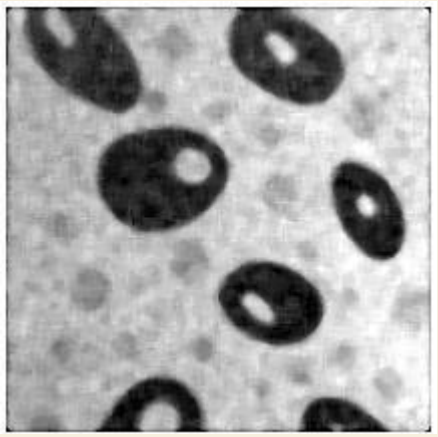


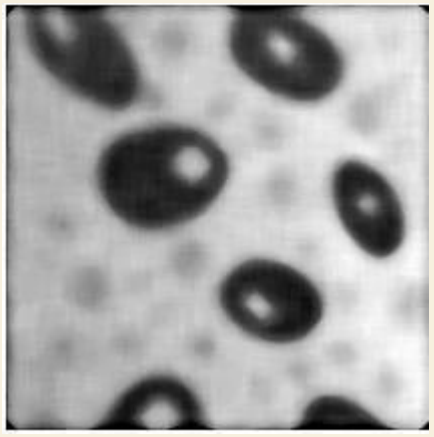
ГАУССОВ ШУМ

Зависит от зашумлённости









БИНАРНОЕ ИЗОБРАЖЕНИЕ

Зависит от размера апертуры

ГАУССОВ ШУМ

- Ранговый фильтр
- Упорядочиваем точки в *n*-окрестности

173	164	170
150	176	169
168	182	166

- (150, 164, 166, 168, 169, 170, 173, 176, 182) – в середине списка

ГАУССОВ ШУМ

- Взвешенный фильтр
- Суммируем с весами

173	164	170
150	176	169
168	182	166

1	1	1
1	1	1
1	1	1

- $\Sigma 167{,}33 \approx 167$

ГАУССОВ ШУМ

- Взвешенный фильтр
- Суммируем с весами

173	164	170
150	176	169
168	182	166

1	2	1
2	4	2
1	2	1

- $\Sigma 169{,}43 \approx 169$

ФИЛЬТРЫ В OPENCV

- Медианный фильтр

void medianBlur (cv::Mat src, CV::Mat dst, int ksize)

src – исходное изображение

dst – результат

ksize – размер фильтра

- Бокс-фильтр

void boxFilter (cv::Mat src, CV::Mat dst, int ksize)

- Гауссов фильтр

void gaussianBlur (cv::Mat src, CV::Mat dst, int ksize)

РЕЗУЛЬТАТ ФИЛЬТРАЦИИ

ОРИГИНАЛ

БОКС-ФИЛЬТР

РЕЗУЛЬТАТ ФИЛЬТРАЦИИ

ОРИГИНАЛ

СВЁРТКА С ГАУССИАНОМ

РЕЗУЛЬТАТ ФИЛЬТРАЦИИ

ОРИГИНАЛ

МЕДИАННЫЙ ФИЛЬТР